Docket No. 211316US2

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Seiji HAYASHIDA
GAU:
SERIAL NO: NEW APPLICATION
EXAMINER:
FILED: Herewith
FOR: COMPILER AND METHOD FOR COMPILING EASILY ADAPTABLE TO PROCESSOR SPECIFICATIONS

REQUEST FOR PRIORITY

ASSISTANT COMMISSIONER FOR PATENTS
WASHINGTON, D.C. 20231

SIR:

☐ Full benefit of the filing date of U.S. Application Serial Number, filed, is claimed pursuant to the provisions of **35 U.S.C. §120**.

☐ Full benefit of the filing date of U.S. Provisional Application Serial Number , filed , is claimed pursuant to the provisions of **35 U.S.C. §119(e)**.

■ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of **35 U.S.C. §119**, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

COUNTRY	APPLICATION NUMBER	MONTH/DAY/YEAR
Japan	2000-210412	July 11, 2000

Certified copies of the corresponding Convention Application(s)

■ are submitted herewith

☐ will be submitted prior to payment of the Final Fee

☐ were filed in prior application Serial No. filed

☐ were submitted to the International Bureau in PCT Application Number .
Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.

☐ (A) Application Serial No.(s) were filed in prior application Serial No. filed ; and

(B) Application Serial No.(s)

☐ are submitted herewith

☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,
MAIER & NEUSTADT, P.C.

Marvin J. Spivak
Registration No. 24,913
C. Irvin McClelland
Registration Number 21,124

22850
Tel. (703) 413-3000
Fax. (703) 413-2220
(OSMMN 10/98)

日　本　国　特　許　庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application: ２０００年　７月１１日

出願番号
Application Number: 特願２０００－２１０４１２

出願人
Applicant(s): 株式会社東芝

CERTIFIED COPY OF
PRIORITY DOCUMENT

２００１年　４月２７日

特許庁長官
Commissioner,
Japan Patent Office 及川耕造

【書類名】　特許願
【整理番号】　4HB003141
【提出日】　平成12年 7月11日
【あて先】　特許庁長官殿
【国際特許分類】　G06F　9/44
【発明の名称】　コンパイラ、コンパイル方法及びコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体
【請求項の数】　9
【発明者】
【住所又は居所】　神奈川県川崎市幸区小向東芝町１番地　株式会社東芝マイクロエレクトロニクスセンター内
【氏名】　林田　聖司
【特許出願人】
【識別番号】　000003078
【氏名又は名称】　株式会社　東芝
【代理人】
【識別番号】　100083806
【弁理士】
【氏名又は名称】　三好　秀和
【電話番号】　03-3504-3075
【選任した代理人】
【識別番号】　100068342
【弁理士】
【氏名又は名称】　三好　保男
【選任した代理人】
【識別番号】　100100712
【弁理士】
【氏名又は名称】　岩▲崎▼　幸邦

【選任した代理人】

【識別番号】　100100929

【弁理士】

【氏名又は名称】　川又　澄雄

【選任した代理人】

【識別番号】　100108707

【弁理士】

【氏名又は名称】　中村　友之

【選任した代理人】

【識別番号】　100095500

【弁理士】

【氏名又は名称】　伊藤　正和

【選任した代理人】

【識別番号】　100101247

【弁理士】

【氏名又は名称】　高橋　俊一

【選任した代理人】

【識別番号】　100098327

【弁理士】

【氏名又は名称】　高松　俊雄

【手数料の表示】

【予納台帳番号】　001982

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　コンパイラ、コンパイル方法及びコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体

【特許請求の範囲】

【請求項１】　ソースプログラム中あるいはヘッダファイル中の組込関数の定義及び属性を読み取り、前記組込関数の定義及び属性に基づいて前記ソースプログラムを機械語に変換することを特徴とするコンパイラ。

【請求項２】　ソースプログラムを入力としオブジェクトコードを生成するコンパイラであって、

前記ソースプログラム内に記述された命令をトークンに分割する字句解析部と、

前記トークンが文法に適しているかを解析し、これらトークンの組み合わせが組込関数及び属性を定義しているかを解析する構文解析部と、

前記構文解析部で解析された組込関数の定義及び属性を記憶する組込関数定義属性記憶手段と、

前記組込関数の定義及び属性に基づいて、前記ソースプログラムを機械語へ変換するコード生成部とを有することを特徴とするコンパイラ。

【請求項３】　前記組込関数の定義に、仮引数型と識別指名が含まれることを特徴とする請求項１又は２記載のコンパイラ。

【請求項４】　コンパイラの起動時または動作中に、組込み関数の定義が開始する旨の宣言を検出する工程と、

前記組込み関数の定義を登録する工程と、

前記組込み関数の属性に関する記述が開始する旨の宣言を検出する工程と、

前記組込み関数の属性を登録する工程と

を少なくとも含むことを特徴とするコンパイル方法。

【請求項５】　コンパイラの起動時または動作中に、組込み関数の定義及び組込み関数の属性に関する記述が開始する旨の宣言を検出する工程と、

前記組込み関数の定義を登録する工程と、

前記組込み関数の属性を登録する工程と

を少なくとも含むことを特徴とするコンパイル方法。

【請求項６】　前記組込み関数の定義に、仮引数の型、識別子名が含まれることを特徴とする請求項４又は５記載のコンパイル方法。

【請求項７】　コンパイラの起動時または動作中に、組込み関数の定義が開始する旨の宣言を検出するステップと、

前記組込み関数の定義を登録するステップと、

前記組込み関数の属性に関する記述が開始する旨の宣言を検出するステップと、

前記組込み関数の属性を登録するステップと

を少なくとも含むことを特徴とするコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項８】　コンパイラの起動時または動作中に、組込み関数の定義及び組込み関数の属性に関する記述が開始する旨の宣言を検出するステップと、

前記組込み関数の定義を登録するステップと、

前記組込み関数の属性を登録するステップと

を少なくとも含むことを特徴とするコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項９】　前記組込み関数の定義に、仮引数の型、識別子名が含まれることを特徴とする請求項７又は８記載のコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、Ｃ言語などの高級言語を機械語に変換（コンパイル）するコンパイラ、コンパイル方法及びコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体に関し、特に組込関数（intrinsics関数）をコンパイラの起動時又は動作中に任意に追加定義可能とする技術に関する。

【０００２】

【従来の技術】

従来、例えば、mov命令（mov Rn, Rm ：レジスタRmの内容をレジスタRnに代入する）に対応する回路構成をプロセッサが有している場合でも、コンパイラがmov命令に対応していない場合には、ユーザであるプログラム作成者はmov命令を使った高級言語によるプログラムを作成することができない。このような場合、プログラム作成者は使用するプロセッサの構造がmov命令に対応しているにも関わらず、"mov Rn, Rm"と記述する代わりに、同機能を表すためにはlw(load word), sw(store word)命令を用いてプログラムを記述しなければならない。すなわち、プロセッサはmov命令に対応する回路構成を持ちながら、lw,swなどの複数の命令を実行しなければならない。

【０００３】

また、単一命令で複数データを並列処理するマルチメディア命令のように、既存命令の組み合わせでは実現できないような命令に対応していないコンパイラを使用した場合には、プログラム作成者はプロセッサ特有の機能を利用することができない。

【０００４】

従って、プロセッサ提供側は、ユーザがプロセッサを効率よく動作可能とするために、プロセッサ特有の仕様（プロセッサ・アーキテクチャおよび命令セット）に対応したコンパイラを提供しなければならない。

【０００５】

プロセッサ特有の仕様（プロセッサ・アーキテクチャおよび命令セット）にコンパイラを対応させる手法としては、intrinsics関数（プログラム言語の処理系が予め用意している関数であり、対象となるプロセッサ特有の高級言語による命令記述を機械語にコンパイルさせるための組込関数）を用いる手法が存在する。

【０００６】

プロセッサ及びコンパイラの提供側は、対象となるプロセッサに応じてintrinsics関数をコンパイラ内部に予め組み込んでおくことにより、プログラム作成者がプロセッサ特有の命令動作を高級言語にて記述することを可能にし、機械語にコンパイルすることを可能にしている。

【０００７】

従来、intrinsics関数はコンパイラ内部に組み込み済みであるため、ユーザが独自にintrinsics関数を定義することができなかった。仕様（プロセッサ・アーキテクチャおよび命令セット）が変化しないプロセッサ用のコンパイラにおいては、この方法で問題ない。

【０００８】

【発明が解決しようとする課題】

しかしながら、ユーザによる仕様（プロセッサ・アーキテクチャおよび命令セット）拡張可能なプロセッサ用のコンパイラでは、ユーザの仕様（プロセッサ・アーキテクチャおよび命令セット）拡張に応じた専用のコンパイラが必要となることから、ユーザが独自にintrinsics関数をカスタマイズできる必要がある。

【０００９】

そこで、本発明は、intrinsics関数をコンパイラの起動時および動作中に任意に追加定義することを可能にするコンパイラ、コンパイル方法及びコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体を提供することを目的とする。

【００１０】

【課題を解決するための手段】

上記課題を解決するため、本発明の特徴は、ソースプログラム中あるいはヘッダファイル中の組込関数の定義及び属性を読み取り、前記組込関数の定義及び属性に基づいて前記ソースプログラムを機械語に変換することにある。

【００１１】

また、本発明の他の特徴は、ソースプログラムを入力としオブジェクトコードを生成するコンパイラであって、前記ソースプログラム内に記述された命令をトークンに分割する字句解析部と、前記トークンが文法に適しているかを解析し、これらトークンの組み合わせが組込関数及び属性を定義しているかを解析する構文解析部と、前記構文解析部で解析された組込関数の定義及び属性を記憶する組込関数定義属性記憶手段と、前記組込関数の定義及び属性に基づいて、前記ソースプログラムを機械語へ変換するコード生成部とを有することにある。

【００１２】

また、組込関数の定義には、仮引数型と識別子名が含まれる。

【００１３】

本発明の特徴によれば、ソースプログラム中の組込関数（intrinsics関数）の定義と属性を解析して、それを記憶手段に記憶することによって、コンパイラの起動時及び動作時にintrinsics関数を任意に追加定義可能となる。

【００１４】

【発明の実施の形態】

以下、図面に基づいて本発明の実施の形態を説明する。以下の図面の記載において、同一又は類似の部分には同一又は類似の符号が付してある。

【００１５】

（１）第１実施形態

本実施形態では、mov Rn,Rm（機能：レジスタRmの内容をレジスタRnに代入する。コードサイズ：２バイト）という命令に関してintrinsics関数を定義する。以下、定義方法、使用方法及びコンパイラでの実装方法に分けて記述する。

【００１６】

（１－１）intrinsics関数の定義方法

独自の予約語（_asm,_reg_dest,_reg_src）と#pragma customを使用して、intrinsics関数を定義する。mov Rn, Rmの定義は、以下のように記述される。

【００１７】

```
void_asm mov (int_reg_dest,int_reg_src) ;
```

この定義方法は、_asmという独自キーワードを追加した以外には、ＩＳＯ／ＪＩＳ　C言語規格（ＩＳＯ／ＩＥＣ９８９９：１９９０）の関数宣言に従っている。この_asmを付加することにより、通常の関数宣言ではなく、intrinsics関数の定義であることを示している。関数名movにより、mov命令のintrinsics関数の定義であることを示す。_reg_dest,_reg_srcは、仮引数の識別子名であるので、ＩＳＯ／ＪＩＳ　C言語規格上は、任意の識別子名が記述できる。仮引数は関数の定義で使われ、それぞれが呼び出されるときに実引数で置き換えられる。intrinsics関数を定義する場合は、これらの特別な予約語を用いることにより、mov命令の動作についての情報をコンパイラに伝達する。_reg_destはレジスタのデ

スティネーションのオペランド、_reg_srcはレジスタのソースのオペランドであることを示す。本実施形態では、_asm, _reg_dest, _reg_srcを予約語として使用したが、ＩＳＯ／ＪＩＳ　Ｃ言語規格の予約語以外の語句であれば、他の語句をintrinsics関数定義用予約語として使用することができる。

【００１８】

次に、#pragma customは、mov命令の場合は、以下のようになる。

【００１９】

#pragma custom mov 2byte

#pragmaは、ＩＳＯ／ＪＩＳ　Ｃ言語規格に規定されている前処理指令のひとつであり、処理系によって独自の拡張が許されている。本実施形態では、#pragma customによって、intrinsics関数の色々な属性の定義を行う。この記述では、mov命令の命令長が２バイトであることを定義している。本実施形態では、customという語句を使用したが、これは任意の語句で置き換え可能である。

【００２０】

（１－２）intrinsics関数の使用方法

下記のＣ言語プログラムのように、intrinsics関数を利用する前に、intrinsics関数の定義を行う。そして、通常の関数呼び出しの記述方法で、intrinsics関数を利用する。

【００２１】

＜Ｃ言語プログラム＞

```
/*intrinsics関数の定義*/
void_asm mov(int_reg_dest,int_reg_src);
#pragma custom mov 2byte
void test(){
  int a,b;
  /*intrinsics関数の利用*/
  mov(a,b);
  }
```

コンパイラは、予め定義されたintrinsics関数の定義内容に従って、コンパイ

ルを行い、以下のようなアセンブラファイルを生成する。指定されたとおり、mov命令が生成されている。

【００２２】

＜生成されたアセンブラファイル＞

```
_test:
  mov$1,$0
  ret
```

（１－３）intrinsics関数の実装方法

図１及び図２にコンパイラの処理の流れを示す。コンパイラは、高級言語で記述されたソースプログラムを入力とし、オブジェクトコードを出力とするソフトウェアプログラムである。

【００２３】

図１は中間コードを生成するコンパイラの構成を示す図である。図１に示すように、コンパイラ１０ａは、まず字句解析部１１による字句解析を行う。「字句解析」では、ソースプログラム１に記述された命令文を意味のある最小の単位であるトークンに分割する処理を行う。

【００２４】

次に、構文解析部１２による構文解析を行う。「構文解析」では、直前の字句解析によって変数や記号など意味のある最小単位であるトークンに分割された命令文が、それぞれの言語で定められた文法に適しているかどうかをチェックする。

【００２５】

そして、中間コード生成部１３による中間コードの生成、中間コード最適化部１４による中間コードの最適化を経て、コード生成部１５ａによるコード生成を行う。「コード生成」では、それまでに行われたソースプログラム１の最小単位への分割、構文エラーのチェックなどの結果を受けて、コード・ジェネレータの機能を用いてソースプログラム１を、アセンブラ形式や２進数形式などの機械語へ変換する処理が行われる。

【００２６】

さらに、コード最適化部１６ａによるコード最適化を行う。「コード最適化」では、直前のコード生成部により変換された機械語に、実際の処理効率を向上させるための変更が加えられる。

【００２７】

そして、コード出力部１７ａからオブジェクトコード３を生成する。

【００２８】

コンパイラ１０ａが中間コード生成を行っているのは、構文解析の直後にコード生成をすると、生成されるプログラム・サイズが非常に大きくなり、変換処理が効率よく行われない場合があるため、ソースプログラム１ａと等価で、しかも簡潔な言語である中間コードに変換してから翻訳処理を行うからである。中間コード生成及び中間コード最適化を省略してもオブジェクトコードは得られる。

【００２９】

図２は中間コードを生成しないコンパイラの構成を示す図である。図２に示すように、コンパイラ１０ｂは、字句解析部１１、構文解析部１２、コード生成部１５ｂ、コード最適化部１６ｂ、コード出力部１７ｂを経て、ソースプログラム１ａからオブジェクトコード３を生成する。

【００３０】

図１又は図２に示すいずれのコンパイラであっても、「構文解析」において、intrinsics関数の定義、使用を識別する必要がある。このための処理の流れを図３に示す。また、属性情報処理の流れを図４に示す。

【００３１】

図３は、intrinsics関数の定義の構文解析処理の流れを示す図である。図３に示すように、まず"_asm"が付加されているかをチェックする（ステップＳ３０）。"_asm"が付加されていなければ、通常の関数の宣言であると認識する（ステップＳ３１）。"_asm"が付加されている場合は、intrinsics関数の定義の処理をする（ステップＳ３２）。まず、仮引数の型情報、識別子の名前の解釈を行う（ステップＳ３３）。仮引数の型の指定方法等に間違いがある場合は、エラーメッセージを出力する（ステップＳ３４）。仮引数の型の指定方法等に間違いが無い場合は、intrinsics関数の定義内容を記号表へ登録する（ステップＳ３５）。記号

表とは、定義したintrinsics関数やその引数をサーチするために使用する表である。

【００３２】

図４は、属性情報処理の流れを示す図である。図４に示すように、"#pragma"の次の字句が"custom"かチェックする（ステップＳ４０）。"custom"でない場合は、他の#pragma指令の処理を行う（ステップＳ４１）。"custom"である場合は、"#pragma custom"の処理を行う（ステップＳ４２）。指定された識別子（例えば、mov）が、intrinsics関数として記号表に登録されているかチェックする（ステップＳ４３）。記号表に登録されていない場合は、エラーメッセージを出力する（ステップＳ４４）。記号表に登録されている場合は、指定された属性情報の解釈を行う（ステップＳ４５）。指定された属性（字句）が解釈できない場合は、エラーメッセージを出力する（ステップＳ４６）。指定された属性（字句、例えば2byte）が解釈できる場合は、intrinsics関数の情報に属性情報を付加する（ステップＳ４７）。

【００３３】

本第１実施形態によれば、"_asm"が付加されているとintrinsics関数の定義の処理が開始され、仮引数の型情報や識別子名に誤りがなければ、intrinsics関数の情報として記号表に登録される。また、"#pragma"の次の字句が"custom"であれば、"#pragma custom"の処理が開始され、指定された識別子が記号表に登録されていれば、指定された情報(例えば、2byte)が属性情報として登録される。これにより、intrinsics関数がコンパイラの起動時及び動作中に定義可能となる。

【００３４】

図５に、intrinsics関数を使用した場合のコンパイラの処理の流れを示す。図５に示すように、関数コールの処理が開始されると、まず使用している関数名を記号表から探す（ステップＳ５０）。そして、記号表に登録されているか、intrinsics関数かをチェックする（ステップＳ５１）。記号表に登録されているが、intrinsics関数ではない場合は、通常の関数の処理を行う（ステップＳ５２）。記号表に登録されているintrinsics関数である場合は、指定された引数（実引数）とintrinsics関数宣言時の引数（仮引数）の情報をチェックする（ステップＳ

５３）。さらに、実引数と仮引数で、数及び型が一致するかをチェックする（ステップＳ５４）。数又は型が一致しない場合は、エラーメッセージを出力する（ステップＳ５５）。数及び型が一致する場合は、intrinsics関数としてコード（中間コード又はアセンブラコード）を生成する（ステップＳ５６）。

【００３５】

これによってユーザが定義したintrinsics関数を用いてコンパイルすることが可能となる。

【００３６】

（２）第２実施形態

第１実施形態ではmov命令の実装方法を説明したが、第２実施形態では汎用的に使用する場合について述べる。

【００３７】

（２－１－１）仮引数の識別子に用いる予約語

第１実施形態で示した"_reg_src"と"_reg_dest"以外に、"_reg_modify","_mem_read","_mem_write","_mem_modify","_imm","_label"を独自の予約語として追加することにより、代表的な機械命令はintrinsics関数として記述可能となる。

【００３８】

"_reg_modify"は、ソースであり、かつデスティネーションでもあるレジスタのオペランドを示す。"_mem_read"はメモリ・リードのオペランドを示す。"_mem_write"はメモリ・ライトのオペランドを示す。"_mem_modify"はメモリ・リードと演算結果のメモリ・ライトのオペランドを示す。"_imm"は即値のオペランドを示す。"_label"は分岐先のラベル名を指定するオペランドを示す。

【００３９】

（２－１－２）#pragma customで指定できる属性

第１実施形態で示した"mov"の属性(2byte)以外に、"read","write","modify","interlock","freeze","uncondition","call","return"の属性を定義することにより、代表的な機械命令が記述可能となる。"read"はintrinsics関数の定義で指定したオペランド以外に、ソースとして使用するオペランドを指定する。"write"はintrinsics関数の定義で指定したオペランド以外に、デスティネーションと

して使用するオペランドを指定する。"modify"はintrinsics関数の定義で指定したオペランド以外に、ソースとデスティネーションとして使用するオペランドを指定する。

【００４０】

たとえば、東芝マイクロプロセッサＴＸ３９のmul命令の仕様では、mul命令は指定したオペランド以外にlowレジスタ($lo)をデスティネーションとする。この場合には、"#pragma custom mul write $lo"と定義する。

【００４１】

"read","write","modify"を指定することにより、intrinsics関数が使用する資源の全ての情報を設定することができ、コンパイラによる機械命令の生成が可能となる。

【００４２】

"interlock"は、コンパイラに命令スケジューリングの必要性を指示する。連続する命令（例えば、命令Ａと命令Ｂ）が、共通のオペランドを使用するためにパイプラインがストールしてしまう場合に、ストールの原因となっているオペランドを使用しない命令（例えば、命令Ｃ）を、共通のオペランドを使用する連続する命令（命令Ａと命令Ｂ）の間に挿入することにより、ストールを防止する。

【００４３】

"freeze"は、このintrinsics関数の前後の命令の順番を変更しないことをコンパイラに指示する。

【００４４】

オペランドとして"_label"を指定した場合、その命令が条件分岐命令か、無条件分岐命令か判別できない。このため、条件分岐命令を既定値(default)とし、無条件分岐命令を指定する場合には"uncondition"を属性として指定する。無条件分岐命令を既定値(default)とし、条件分岐命令を指定する場合には"condition"を属性として指定する。

【００４５】

"return"は、関数コールからの復帰命令をintrinsics関数として定義した場合に使用する。

【００４６】

（２－２）次に、コプロセッサ命令を定義する方法を説明する。Ｃ言語プログラムは以下のように記述する。

【００４７】

```
void_asm cpmov (int _cop_dest,int _cop_src);
#pragma custom cpmov cop 2byte
```

コプロセッサ命令であることを示すために、#pragma customの属性に"cop"を追加する。これは、指定した関数（例えば、"cpmov"）が、コプロセッサ命令であることを示す。

【００４８】

また、仮引数の識別子に用いる予約語として、"_cop_src","_cop_dest","_cop_modify"を追加する。"_cop_src"は、コプロセッサ・レジスタのソース・オペランドであることを示す。"_cop_dest"はコプロセッサ・レジスタのデスティネーション・オペランドであることを示す。"_cop_modify"はコプロセッサ・レジスタのソース・オペランドでありかつデスティネーション・オペランドであることを示す。

【００４９】

（２－３）次に、ＶＬＩＷ（Very Long Instruction Word）やスーパースカラ型のプロセッサのコンパイラのために、intrinsics関数に対してスケジューリング属性を設定する方法を説明する。ＶＬＩＷやスーパースカラでは、同時に演算できる命令に制限がある。このため、各命令に対して、どの演算器で実行できるかを設定する必要がある。この情報を#pragma customの属性として設定することにより、ＶＬＩＷやスーパースカラ型のプロセッサに対応することができる。

【００５０】

（２－４）また、intrinsics関数の第１引数が、"_reg_dest"または"_cop_dest"の場合は、関数の返却値として、設定することができる。このとき第２引数以降が存在する場合は、その第２引数以降は一つずつ繰り上がる。例えば、

```
void _asm mov(int _reg_dest,int _reg_src);は、
int _asm mov(int _reg_src);
```

と同様の内容である。この場合、代入先を"="演算子で指定することができるようになり、Ｃ言語らしくなるため、可読性が向上する。

【００５１】

第１実施形態と同様の内容のプログラムが、以下のように記述される。

【００５２】

```
int _asm mov(int _reg_src);
#pragma custom mov 2byte
void test(){
      int a,b;
       a = mov(b);
       }
```

本第２実施形態によれば、コプロセッサ、ＶＬＩＷ又はスーパースカラ型プロセッサのintrinsics関数がコンパイラの起動時及び動作中に定義可能となる。

【００５３】

また、代入先を"="演算子で指定することができるようになり、Ｃ言語らしくなるため、可読性が向上する。

【００５４】

さらに、（１）ある命令（例えば、mov命令）が、「通常の関数」として実装されている場合と、機械命令として実装されているので「intrinsics関数」として定義される場合、の両方に対応するために、「intrinsics関数の引数と返り値の形式」を「通常の関数の引数と返り値の形式」と同じにする、（２）コンパイラが持っている条件コンパイルを利用し、起動時に指定するコマンドライン・オプションによって、「intrinsics関数」として定義するか、「通常の関数」として定義するかを選択する、ことにより、ソースプログラムを記述する際に、ある命令（例えば、mov命令）が「通常の関数」として実装されているか、「intrinsics関数」として定義されているかを、プログラマーが意識する必要がなくなる。このため、プログラミングが容易となり、作成したプログラムの再利用性も向上する。

【００５５】

なお、条件コンパイルとは、Cコンパイラでは、#ifdef,#else,#endif として実装されており、コンパイラ起動時に指定する マクロ定義のコマンドライン・オプションによって、コンパイルする内容を変更することができる機能のことである。

【００５６】

条件コンパイルの例を以下に示す。

【００５７】

```
#ifdef __USE_MOV__  /* マクロ __USE_MOV__ が定義された場合の処理 */
/* intrinsics関数 */
int __asm mov(int __reg_src);
#pragma custom mov 2byte
#else /* マクロ __USE_MOV__ が定義されない場合の処理 */
/* 通常の関数 */
int mov(int src); /* この関数は、mov命令と同様の処理を行う */
#endif
/* 実際の使用 */
void test() {
 int i,j;
 /* intrinsics関数がどうかに関係なく使用できる */
 i = mov(j);
 }
```

（３）第３実施形態

第１、２実施形態においては#pragma customも用いて定義したが、第３実施形態では#pragma customを用いないで定義する方法を説明する。

【００５８】

#pragma customで指定した属性を関数の定義中に含める場合は、さらに独自の予約語を追加することになる。この場合、複数の予約語を追加する方法と１つの予約語を追加する方法とがある。

【００５９】

（３－１）複数の予約語を追加する場合

属性毎に、予約語(_2byte,_read,_write,_modify,_interlock,_freeze,_uncondition,_call,_return等)を追加することにより、#pragma customがなくても属性を設定することが可能となる。例えば、第１実施形態のmov命令は、

void_asm_2byte mov(int_reg_dest,int_reg_src);

と定義することができる。

【００６０】

（３－２）１つの予約語を追加する場合

属性毎に予約語を追加するのでなく、１つの予約語を追加することによって、定義することもできる。"_attr"という予約語を定義した場合、第１実施形態のmov命令は、

void_asm_attr(2byte) mov(int_reg_dest,int_reg_src);

と定義することができる。この場合、"_attr"の構文分析時に、指定された属性をチェックする。予約語が少ない方が、プログラミング上の制約が少なくなるため好ましい。

【００６１】

なお、上記第１～３実施形態ではintrinsics関数に必要な情報を、コンパイルするソースファイル中に記述したが、別ファイルからintrinsics関数に必要な情報を読み込むことも可能である。図６に、intrinsics関数の情報を別ファイルとする場合の処理の流れの概略を示す。図６に示すように、ソースファイル１ｂとintrinsics関数情報ファイル６０をコンパイラ１０に入力し、オブジェクトコード３を得る。

【００６２】

この場合は、第１～３実施形態で示したような予約語などに拘束されることはないので、intrinsics関数に必要な情報を任意の書式で記述することが可能となる。

【００６３】

【発明の効果】

上記の如く、本発明によれば、intrinsics関数の動作に必要な情報をコンパイ

ラの外部から設定できるようにすることにより、コンパイラを容易に拡張することができる。このため、ユーザがコンパイラをカスタマイズすることもできる。

【００６４】

また、intrinsics関数の定義を、コンパイルの対象であるソースファイル内に混在させることにより、以下の効果も得られる。

【００６５】

１）intrinsics関数の定義は、通常の関数定義にいくつかの予約語を追加しただけなので、ユーザが容易に理解することができる。

【００６６】

２）ユーザに、Ｃ言語のヘッダファイルとして提供できるため、リリースが容易になる。

【００６７】

３）intrinsics関数の第１引数が、ディスティネーションの場合は、関数定義を変更することにより、ソースファイル中のintrinsics関数名（例えば、mov関数）と同じ名称の命令（例えば、mov命令）を使用する部分を変更しなくてすむ。これにより、プログラムの再利用性が向上する。

【図面の簡単な説明】

【図１】

中間コードを生成するコンパイラの構成を示す図である。

【図２】

中間コードを生成しないコンパイラの構成を示す図である。

【図３】

intrinsics関数の定義処理の流れを示す図である。

【図４】

属性情報処理の流れを示す図である。

【図５】

intrinsics関数を使用した場合のコンパイラの処理の流れを示す図である。

【図６】

intrinsics関数の情報を別ファイルとする場合の処理の流れの概略を示す図で

ある。

【符号の説明】

１　ソースプログラム

３　オブジェクトコード

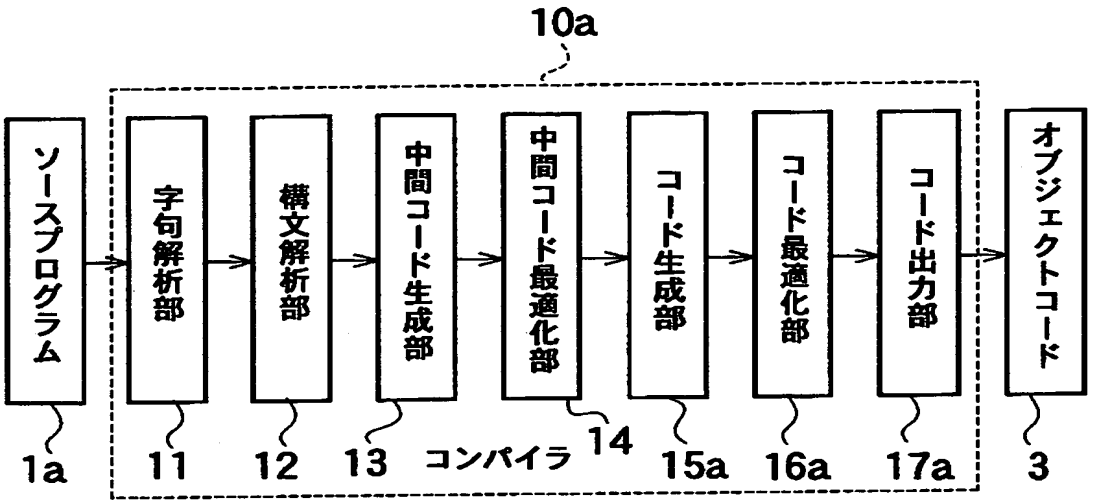
１０　コンパイラ

１１　字句解析部

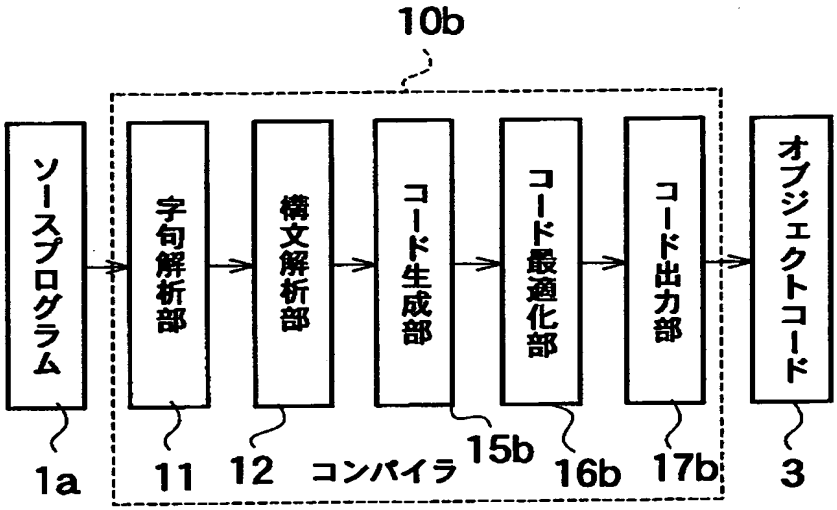
１２　構文解析部

【書類名】　　図面

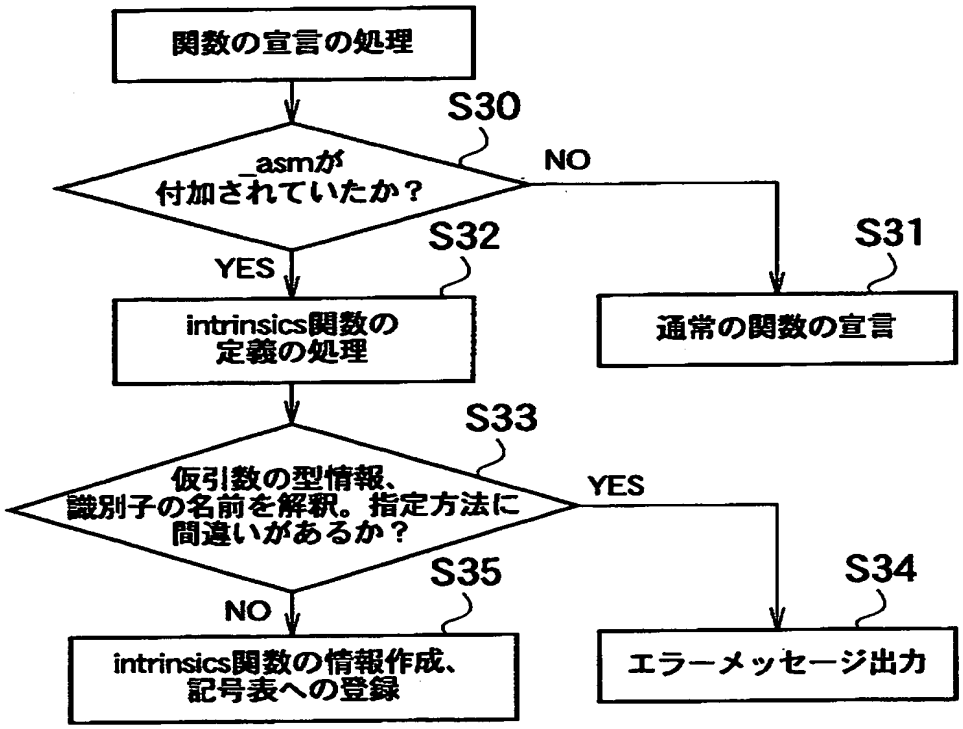
【図１】


10a
ソースプログラム
字句解析部
構文解析部
中間コード生成部
中間コード最適化部
コード生成部
コード最適化部
コード出力部
オブジェクトコード
1a
11
12
13
コンパイラ
14
15a
16a
17a
3

【図２】

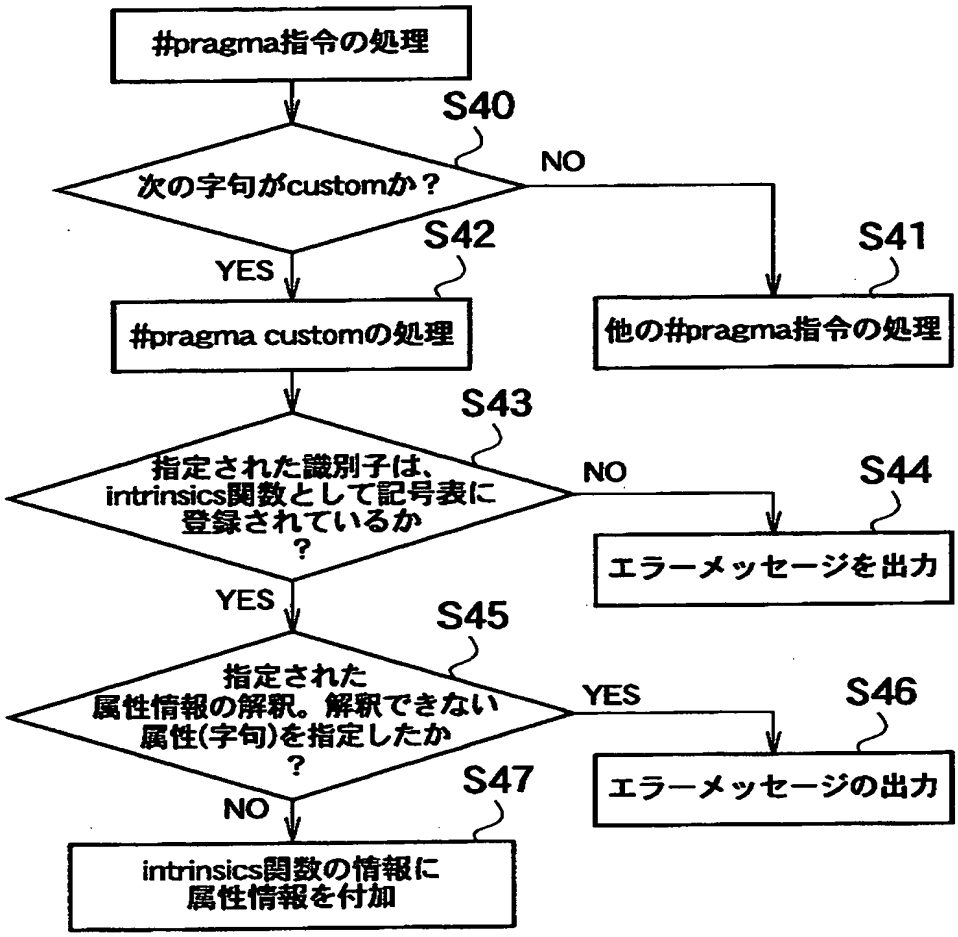

10b
ソースプログラム
字句解析部
構文解析部
コード生成部
コード最適化部
コード出力部
オブジェクトコード
1a
11
12
コンパイラ
15b
16b
17b
3

【図３】

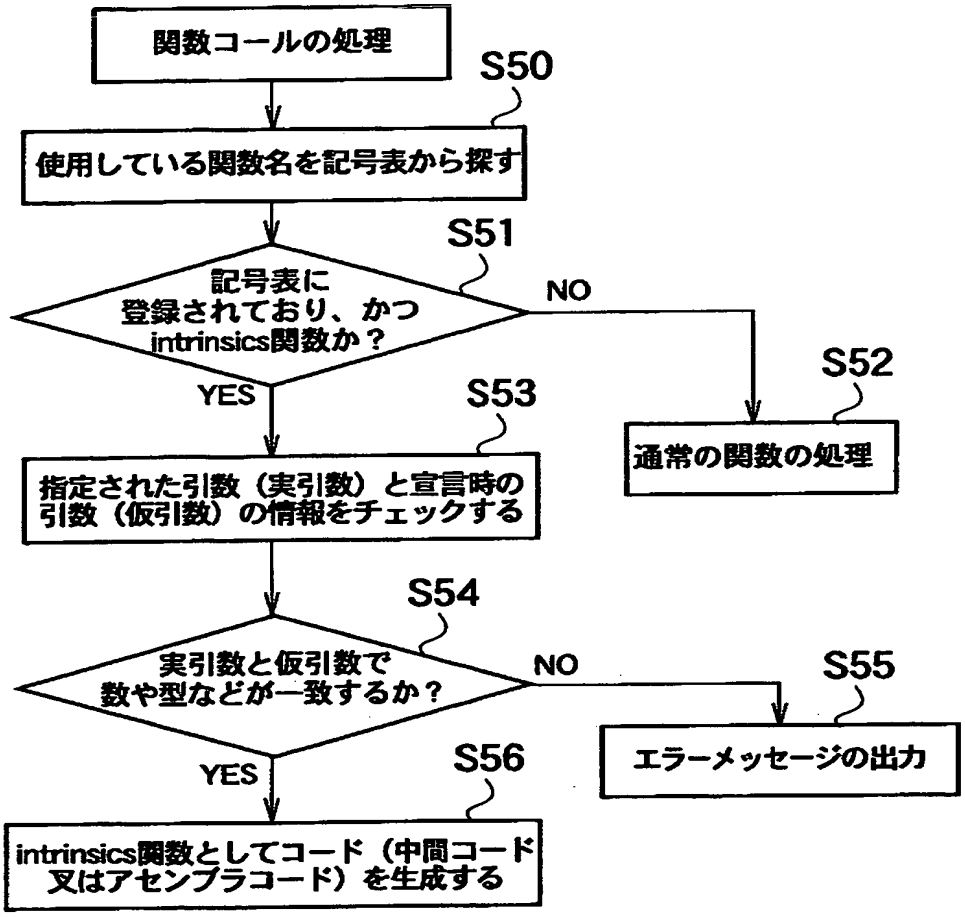


出証特２００１－３０３５９７１

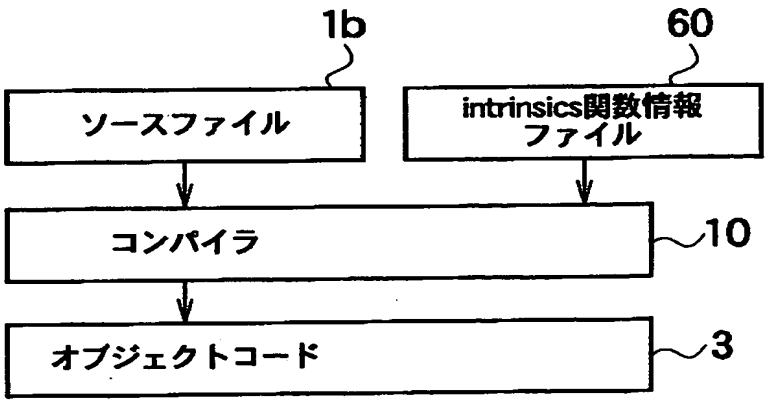
【図４】


#pragma指令の処理
S40
次の字句がcustomか？
NO
S41
他の#pragma指令の処理
YES
S42
#pragma customの処理
S43
指定された識別子は、intrinsics関数として記号表に登録されているか？
NO
S44
エラーメッセージを出力
YES
S45
指定された属性情報の解釈。解釈できない属性(字句)を指定したか？
YES
S46
エラーメッセージの出力
NO
S47
intrinsics関数の情報に属性情報を付加

【図５】


関数コールの処理
S50
使用している関数名を記号表から探す
S51
記号表に
登録されており、かつ
intrinsics関数か？
NO
S52
通常の関数の処理
YES
S53
指定された引数（実引数）と宣言時の
引数（仮引数）の情報をチェックする
S54
実引数と仮引数で
数や型などが一致するか？
NO
S55
エラーメッセージの出力
YES
S56
intrinsics関数としてコード（中間コード
又はアセンブラコード）を生成する

【図６】


1b
60
ソースファイル
intrinsics関数情報
ファイル
コンパイラ
10
オブジェクトコード
3

【書類名】　　要約書

【要約】

【課題】　組込関数（intrinsics関数）をコンパイラの起動時又は動作中に定義可能として、コンパイラの拡張を容易にする。

【解決手段】　ソースプログラム１を入力としオブジェクトコード３を生成するコンパイラ１０であって、ソースプログラム１内に記述された命令をトークンに分割する字句解析部１１と、トークンが文法に適しているかを解析する構文解析部１２と、ソースプログラム１を機械語へ変換するコード生成部１５と、組込関数定義属性を記憶する組込関数定義属性記憶手段を有し、構文解析部１２が、組込関数の定義と属性を解析し、組込関数定義属性記憶手段に記憶する。

【選択図】　　図１

出願人履歴情報

識別番号　〔０００００３０７８〕

１．変更年月日　１９９０年　８月２２日
［変更理由］　新規登録
住　所　神奈川県川崎市幸区堀川町７２番地
氏　名　株式会社東芝